(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 629 946 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **94304250.7**

(51) Int. Cl.[5] : **G06F 9/44**

(22) Date of filing : **13.06.94**

(30) Priority : **14.06.93 US 77229**

(43) Date of publication of application :
21.12.94 Bulletin 94/51

(84) Designated Contracting States :
DE FR GB

(71) Applicant : **INTERNATIONAL BUSINESS MACHINES CORPORATION**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : **Danforth, Scott Harrison**
10011 Woodland Village Drive
Austin, Texas 78750 (US)
Inventor : **Forman, Ira Richard**
2100 Cypress Point East
Austin, Texas 78746 (US)
Inventor : **Madduri, Hari Haranath**
7004 Anagua
Austin, Texas 78750 (US)

(74) Representative : **Davies, Simon Robert**
I B M
UK Intellectual Property Department
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(54) System and method for enabling before/after method processing in an object oriented system.

(57) A system for creating before and after behaviour upon invocation of a method in an object-oriented system. The framework provides metaclasses classes containing methods for dispatching a before method and an after method at the time of invocation of each client method in subclass instances. Object-oriented system properties of inheritance and encapsulation are supported as are derived metaclasses. Derivation ensures that the specification syntax for each class does not impact the expected result. The combination of explicit before after classes, dispatcher class, and derived metaclasses ensures that the system will have associative composition.

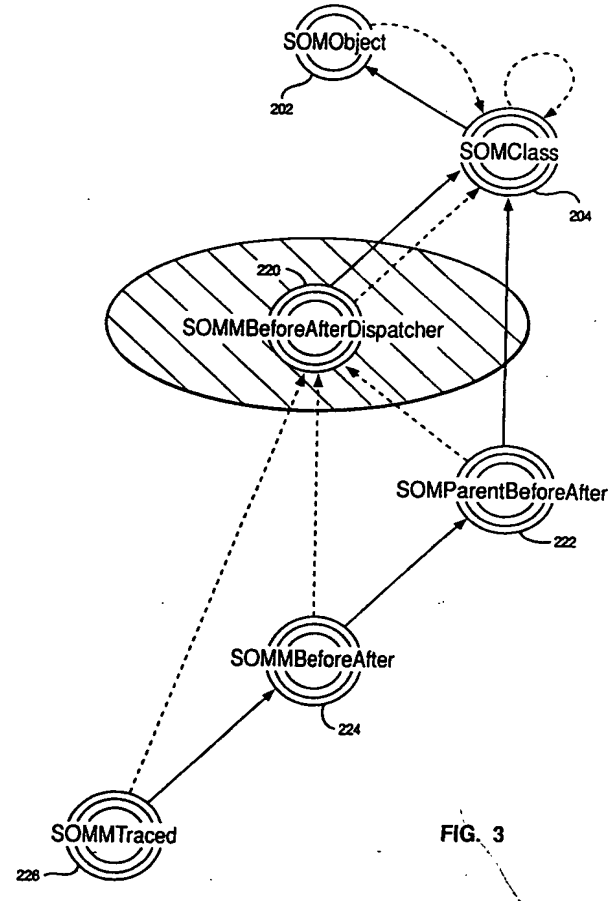

FIG. 3

EP 0 629 946 A2

The present invention relates to electronic data processing systems and more particularly to systems in which processing methods and data are gathered into objects. Still more particularly, the present invention relates to object-oriented systems in which special methods may be processed before and after defined processing methods.

The development of application and system software for data processing systems has traditionally been a time consuming task. The field of software engineering has attempted to overcome the limitations of traditional techniques by proposing new, more efficient software development models. Object-oriented programming has emerged as a promising technology that will allow rapid development, implementation, and customization of new software systems.

Progress in programming technology can be viewed as being shown by the increased level of abstraction employed. Programming technology has progressed through abstractions that grouped ever larger entities. Assembly language instructions were first gathered into control structures, and control structures latter gathered in to procedures. Procedures, in turn, were gathered into abstract data types. Object-oriented programming can be viewed as providing a higher level of abstraction of programming entities than these previous techniques. Object-oriented programming gathers abstract data types into an inheritance hierarchy. The present invention is directed to a further abstraction, a metaclass that provides before and after processing for methods of instances of its instances.

Object-oriented programming uses a toolkit of system objects that can be assembled to perform the required task. Each object has certain data attributes and processes or methods that operate on that data. Data is said to be "encapsulated" by an object and can only be modified by the object methods. Methods are invoked by sending a message to an object identifying the method and supplying any needed arguments.

Object-oriented systems have two important properties in addition to encapsulation. Firstly, "inheritance" which is the ability to derive a new object from an existing object and to inherit all properties, including methods and data structure, from the existing object. The new object may have certain unique features that are supplied as overrides or modifications to the existing class. For example, a new subclass needs to specify only the functions and data that distinguish that class from the existing more general class. Secondly, "polymorphism" which is the ability of an entity (e.g. a variable) to refer at run-time to instances of different classes. Practically, this means that a single message to an object can be processed in different ways depending on the object itself.

Inheritance and polymorphism create a powerful structure for implementing new software systems. The software developer does not have to develop each piece of a system, he or she need only specify the unique features of the system.

The power of object-oriented systems is realized through the development of system "frameworks." A framework is a collection of base classes that can be used by a system implementor to create a final systems product. The framework is defined and developed to work together as a system. Conceptually, the framework is much like a set of standard hardware components used by computer hardware builders. Each of the components has certain defined functions and interfaces and the engineer assembles these components according to a particular design to create a unique hardware system. Object frameworks, like electronic components, typically cannot be changed by the engineer. Instead the engineer must add other objects that interface with the framework in the manner defined by the framework.

Examples of object frameworks are contained in the SOMobjects product developed by the IBM Corp. SOMobjects defines a number of objects, object classes, and object metaclasses for use in software system development. The system developer uses components of the SOMobjects framework and extends those by creating new subclasses based on the framework classes. SOMobjects provides a compiler language neutral interface description language (IDL) for specifying new subclasses and methods. The IDL specification created by the software engineer contains information about the parent classes used by the engineer and specifies any methods that override methods in parent classes.

Computer software developers frequently need to know the state of a computer system before execution of a process and then again after execution of that process. This information can be used when debugging a new system to make sure the process is performing correctly. It can also be used to implement features such as security locks, record locks, and database logging. Existing object-oriented systems do not provide an easy way to specify methods to be executed before and after each instance method without explicitly specifying the behaviour for each instance method.

The technical problem addressed by the present invention is how to provide an object based ability to specify methods to be performed before and after the methods of all instances of a particular class and ensure that these methods compose.

Accordingly, the present invention provides a system for enabling construction of object-oriented classes implementing before and after behaviour for each dispatch of a method process, said system operating in a computer system having memory means and processing means said system including at least one class means having data and methods for processing said data, the system comprising: means for dispatching a BeforeMethod and an AfterMethod re-

spectively before and after dispatching a method process, said means for dispatching being responsive to signalling of a class means; class means for signalling method processing dispatch, said means being responsive to before and after behaviour requests, said class means being a subclass of one of said at least one class means and an instance of said means for dispatching; and subclass means for generating before and after behaviour requests to said class means, said subclass means being a subclass of said class means and being responsive to an instance method invocation.

An embodiment of the invention provides a method for generating associative object metaclass compositions in a computer system having memory and at least one central processing unit and, the method comprising the steps of: specifying a plurality of classes having data and methods for operating on said data; creating a modifiable metaclass for dispatching for execution before or after method invocations; testing said plurality of classes and said modifiable metaclass to determine a set of methods required for consistency; and creating a metaclass hierarchy based on said testing.

A further embodiment of the invention provides a system enabling special method processing before and after processing of a method defined for an object in an object-oriented computer system, each object having one or more object classes and each of said one or more object classes having one or more object metaclasses, the special method processing enabled in an object-oriented framework having unmodifiable and modifiable classes, the system comprising: storage means for storing said objects, classes and metaclasses; dispatch means for dispatching said methods and said special methods, said dispatch means being a modifiable metaclass in said object-oriented framework.

The present invention solves the before and after method execution problem by providing an object-oriented framework that has a BeforeAfterDispatch metaclass as the class for a BeforeAfter class. Classes having before/after behaviours can be created as subclasses of the BeforeAfter class. The BeforeAfterDispatch metaclass supports the dispatch methods necessary to implement before/after processing and is structured to ensure associative composition of subclass descriptions.

An embodiment of the present invention is directed to a system for enabling construction of object-oriented classes implementing before and after behaviour for each dispatch of a method process, the system operating in a computer system having memory means and processing means, the system including at least one class means having data and methods for processing the data. The system has the following components: means for dispatching a BeforeMethod and an AfterMethod respectively before and after dispatching a method process, the means for dispatching being responsive to signalling of a class means; class means for signalling method processing dispatch, the means being responsive to before and after behaviour requests, the class means being a subclass of one of the at least one class means and an instance of the means for dispatching; and subclass means for generating before and after behaviour requests to the class means, the subclass means being a subclass of the BeforeAfter Class and being responsive to an instance method invocation.

The invention advantageously enables before/after processing for all methods of a particular class.

Preferably, an embodiment of the invention further advantageously implements a before/after processing structure that consistently composes at run time.

Preferably, the invention provides a before/after structure that is associative, i.e. composition of three or more beforeAfter metaclasses produces the same result regardless of the order in which they are composed, e.g., ensuring that (A+B)+C = A+(B+C).

An embodiment of the present invention will now be described, by way of examply only, with reference to the accompanying drawings in which:

figure 1 is a block diagram of a computer system of the type to embody the present invention;

figure 2 is a diagram illustrating a basic System Object Model framework and labelling the framework elements;

figure 3 is an object diagram illustrating the preferred embodiment of the present invention;

figure 4 is an object diagram illustrating an application of the framework of the preferred embodiment.

Figure 1 illustrates a computer system capable of being structured according to the present invention. The computer system 100 has a processor 101, a random access memory 104, a permanent storage device 102, a communications adapter 106 connecting the system to a network 108, and an I/O controller 110 controlling the operation of a display 112, a keyboard 114, and a pointing device 116. The computer system can be one of many known systems such as the IBM Personal System/2 (PS/2) computer system or the IBM RISC System/6000 computer system. (IBM, Personal System/2, PS/2, and RISC System/6000 are trademarks of the IBM Corp.) The components can be of any known type. Permanent storage device 104 can be a fixed disk storage system, a removable diskette, or other fixed or removable media such as tape cartridge, CD-ROM, or WORM. The framework of the present invention may be embodied in a removable media for storage and distribution.

The terminology of the IBM System Object Model object framework is presented with reference to Figure 2. The IBM System Object Model product SOMobjects is an object-oriented framework. The framework

consists of classes and objects that belong to those classes. In SOMobjects, classes are themselves objects and have all the properties of an object. Classes have the additional property of containing information about the methods or behaviours to which each object of the class responds. An object responds to a method if the object contains code for that method. A class is said to support a method when objects of that class respond to that method. Methods supported by a class are those explicitly defined for that class and those inherited from classes that are parent classes for that class. Objects respond only to those methods supported by their class. The class of a class is termed a meta-class.

SOMobjects has as its root object SOMObject 202. The class of SOMObject is SOMClass 204. A system developer can use SOMobjects to add additional classes. For example, the class "Dog" 206 is a subclass of SOMObject 202. "Rover" 207 is an instance of "Dog". Two relationships between objects can be expressed. The "instance of" relation is shown in the figure by a dashed arrow, e.g. 208 from an object to its class. The inverse relation is "class of", i.e., Dog is an "instance of" SOMClass and SOMClass is the "class of" Dog. The second relationship is "subclass of" represented by solid lines, e.g. 210, and its inverse "parent of". Thus Dog is a "subclass of" SOMObject and SOMObject is the "parent of" Dog. Subclasses inherit methods from parent classes and can define additional methods that refine the implementation.

SOMClass is identified in Figure 2 as being in the set of metaclasses. This is because it is the "class of" a class. Additional metaclasses, classes and objects can be created using known object-oriented techniques.

The preferred embodiment of the present invention is shown in Figure 3. The framework of Figure 2 has been extended by adding four new metaclasses. SOMMBeforeAfterDispatcher 220 is a subclass of and an instance of SOMClass 204. SOMMParentBeforeAfter 222 is a subclass of SOMClass 204 and an instance of SOMMBeforeAfterDispatcher 220. SOMMBeforeAfter 224 is a subclass of SOMMParentBeforeAfter and an instance of SOMMBeforeAfterDispatcher 220. Finally, SOMMTraced 226 is a subclass of SOMMBeforeAfter 224 and an instance of SOMMBeforeAfterDispatcher 220.

SOMMBeforeAfterDispatcher 220 is the key component of the new framework. This is a class of several metaclasses (and thus a meta-metaclass) and provides BeforeAfter behaviours to its instance metaclasses, e.g., SOMBeforeAfter 224. Any subclass of SOMMBeforeAfter 224 or the other metaclasses will also have the ability to cause execution of a method before each instance method execution and after each instance method execution by specifying a BeforeMethod or an AfterMethod or both. The abstraction of BeforeAfter processing into a metametaclass allows behaviour implementation without specification in the Interface Definition Language (IDL) that is used to describe objects and their interrelationships.

Procedurally, implementation of before/after processing requires that the BeforeMethod be dispatched for execution, the instance method be dispatched for execution, then the AfterMethod be dispatched for execution. This procedure can be embodied in Dispatcher methods to handle dispatching of methods for execution. Because SOMMBeforeAfter 224 and its subclasses must respond to this method, it must be supported by the class of SOMMBeforeAfter, i.e. metaclass SOMMBeforeAfterDispatcher 220. In particular, the SOMMBeforeAfterDispatcher implements BeforeMethodDispatch that searches for and dispatches any BeforeMethod in the class or parent classes, and AfterMethodDispatch that searches for and dispatches any AfterMethod in the class or parent classes.

The above structure provides a framework that allows a software developer to create a before/after class by creating a subclass of SOMMBeforeAfter and by defining a BeforeMethod and an AfterMethod.

SOMMTraced 226 is an example of a before/after class supplied with the SOMobjects framework. SOMMTraced provides the methods to support tracing of method execution in an object-oriented system. For example, it provides a BeforeMethod to print the calling parameters to a method call and an AfterMethod to print the return value from that method call.

SOMMParentBeforeAfter 222 is provided for those situations where before/after processing is desired only for inherited methods.

The structure of the preferred embodiment metametaclass SOMBeforeAfterDipatch is important because it provides associative composition of metaclasses for an object-oriented system. SOMobjects provides a syntax for specifying relationships between objects. This language allows the developer to provide, for example, an explicitly defined metaclass for each defined class. A complex system may be validly expressed in several different ways. Consistent operation of the system requires that each of those valid expressions provide the same result when processed.

SOMobjects enforces consistency by deriving metaclasses for each object. The process for deriving metaclasses is discussed in U.S. Patent Application Serial Number 08/042,930 filed April 5, 1993.

SOMobjects allows multiple inheritance, i.e., a class can be a subclass or instance of two or more classes. Derived metaclasses ensures that each instance of that class validly responds to inherited methods. SOMobjects checks at runtime to determine whether the metaclass for each class supports the methods that could be executed by instances of

that class. If not, SOMobjects derives the metaclass necessary to support all required methods.

The SOMMBeforeAfter and SOMMBeforeAfterDispatcher structure supports metaclass derivation. An example of this is shown in Figure 4. A "Dog" class 231 can be created as a subclass of SOMObject to define the properties of a class of Dogs. Dog may include a method "Fetch" that causes an animated image of a dog to cross the display screen 112. "RinTinTin" 233 can be created as an instance of "Dog" having certain properties. Sending the message "Fetch" to "RinTinTin" would cause an animated dog to cross the display screen.

The developer may want to add before/after features to the class of Dog. This can be done by creating "Barking" 232 a subclass of SOMMBeforeAfter. "Barking" can include a BeforeMethod that causes the word "WOOF" to be printed, and an AfterMethod that causes the word "woof" to be printed. The class "BarkingDog" 234 is created as a subclass of Dog 231 having an explicit metaclass of Barking 232. An instance of BarkingDog, e.g., Lassie, 236 would respond to the message "Fetch" by causing the printing of "WOOF", the animated crossing of the screen by a dog, and the printing of "woof".

A fierce dog could be created in a similar way by specifying subclassing SOMMBeforeAfter to create a "Fierce" class 230. The FierceDog would say "GRRR" before acting and "grrr" afterwards.

The composition problem is noticed when the developer attempts to create a "FierceBarkingDog". This new class can be defined in at least three ways. First, the class could be defined as a subclass of Dog with an explicit metaclass of "FierceBarking". "FierceBarking" would be created by multiple inheritance from Fierce and Barking. Second, the class could be defined as a subclass of FierceDog and BarkingDog. Finally, the class could be defined as a subclass of BarkingDog with an explicit metaclass of Fierce. Each of these represent valid SOMObject expressions. It is essential that each produces the same result.

SOMObject metaclass processing will analyze the methods required to be supported and derive any necessary metaclass. The result will be a use of metaclass FierceBarking as the actual metaclass for each of the three defined FierceBarkingDogs defined above. The explicit metaclass definition would be overridden to ensure consistency. A "Rover" instance of FierceBarkingDog would respond "GRRRWOOF" (animated dog) "woofgrrr" regardless of which of the three classes it instantiates. The result is associative composition because the form of definition is immaterial. Note, however, that the composition is not commutative, i.e., a FierceBarkingDog is not the same as a BarkingFierceDog.

The code for implementing the preferred embodiment of somDispatch in a before/after environment is illustrated below. This code is illustrative only and it will be clear to others in the field that many alternative embodiments exists without departing from the spirit of the invention.

The basic dispatch routine can be implemented as follows. "Self" refers to the instance receiving the message. "clientMethod" is the identifier of the method being invoked by the application and acquiring before/after behaviour.

```
somDispatch(self, clientMethod)
        BeforeMethodDispatch(class(class(self)),
self, ...);
        retval := clientMethod(self, ...);
        AfterMethodDispatch(class(class(self)),
self,...);
        return retval;
```

The BeforeMethodDispatch is implemented as follows:

```
        BeforeMethodDispatch(aMetaclass, clientOb-
ject, ...)
                if aMetaclass defines BeforeMethod
                        BeforeMethod(clientObject)
                else
                        for all parents of aMetaclass that
support BeforeMethod
                                BeforeMethodDispatch(
aParent, clientObject, ...)
```

This method will cause a BeforeMethod to be executed if defined for the client object or will cause a hierarchy traversal towards SOMClass looking for the first metaclass that defines BeforeMethod.

AfterMethodDispatch is implemented as follows:

```
        AfterMethodDispatch(aMetaclass,  clientOb-
ject, ...)
                if aMetaclass defines AfterMethod
                        AfterMethod (clientObject, ...)
                else
                        for all parents of aMetaclass that
support AfterMethod(in reverse order)
                        AfterMethodDispatch(aParent,
clientObject, ...)
```

AfterMethodDispatch traverses the hierarchy in reverse order where an AfterMethod is not defined for the client object.

Claims

1. A system for enabling construction of object-oriented classes implementing before and after behaviour for each dispatch of a method process, said system operating in a computer system having memory means and processing means (101) said system including at least one class means having data and methods for processing said data, the system comprising:

   means (220) for dispatching a BeforeMethod and an AfterMethod respectively before and after dispatching a method process, said means

for dispatching being responsive to signalling of a class means;

class means (224) for signalling method processing dispatch, said means being responsive to before and after behaviour requests, said class means being a subclass of one of said at least one class means and an instance of said means for dispatching; and

subclass means (230) for generating before and after behaviour requests to said class means, said subclass means being a subclass of said class means (224) and being responsive to an instance method invocation.

2. A method for generating associative object metaclass compositions in a computer system having memory (104) and at least one central processing unit (101) and, the method comprising the steps of:

specifying a plurality of classes (202, 231,234) having data and methods for operating on said data;

creating a modifiable metaclass for dispatching for execution before or after method invocations;

testing said plurality of classes (202, 231,234) and said modifiable metaclass to determine a set of methods required for consistency; and

creating a metaclass hierarchy (232,224, 222) based on said testing.

3. A system enabling special method processing before and after processing of a method defined for an object (236) in an object-oriented computer system, each object having one or more object classes (224) and each of said one or more object classes having one or more object metaclasses (232), the special method processing enabled in an object-oriented framework having unmodifiable and modifiable classes, the system comprising:

storage means (104) for storing said objects, classes and metaclasses;

dispatch means (220) for dispatching said methods and said special methods, said dispatch means being a modifiable metaclass in said object-oriented framework.

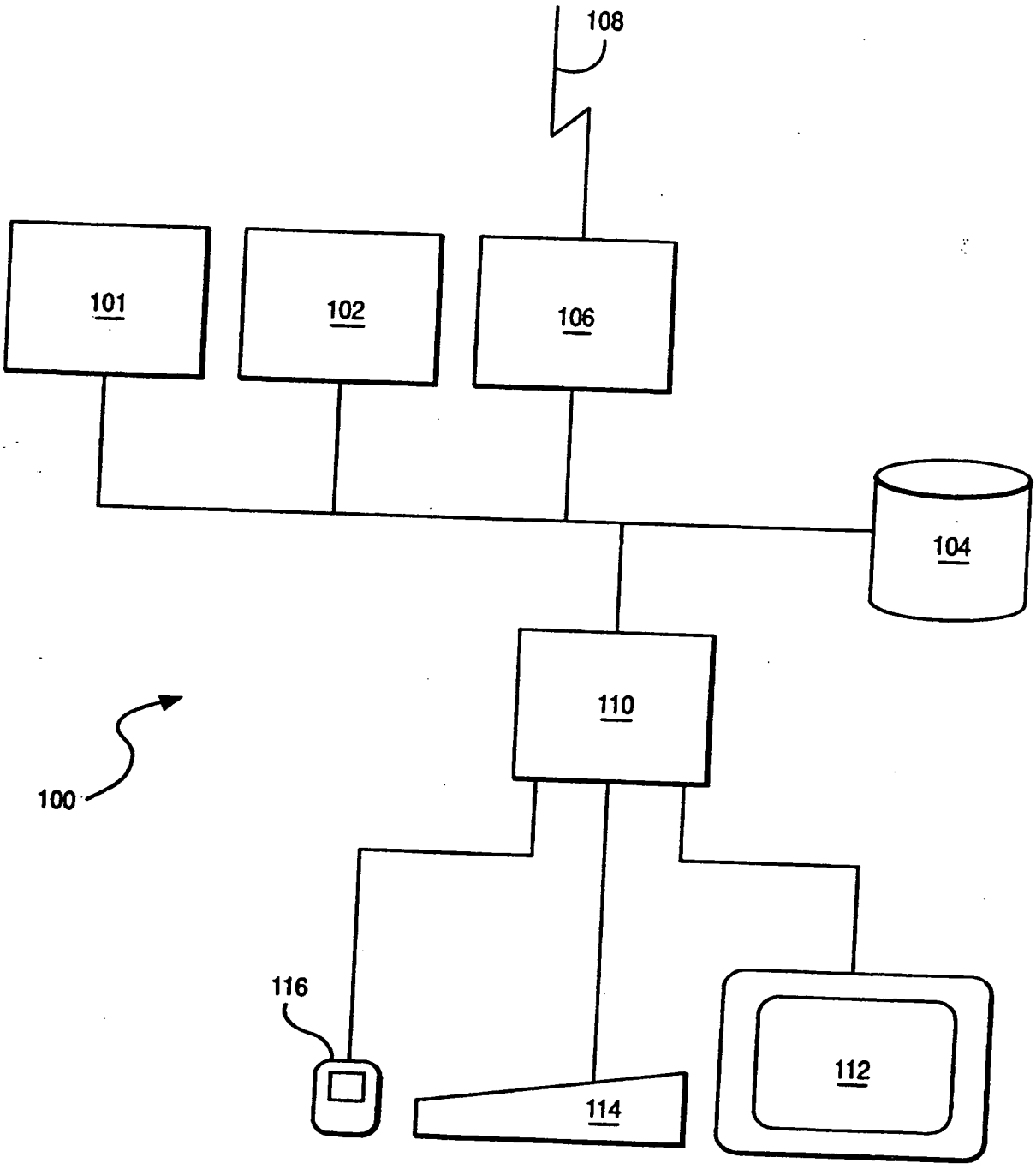

FIG. 1

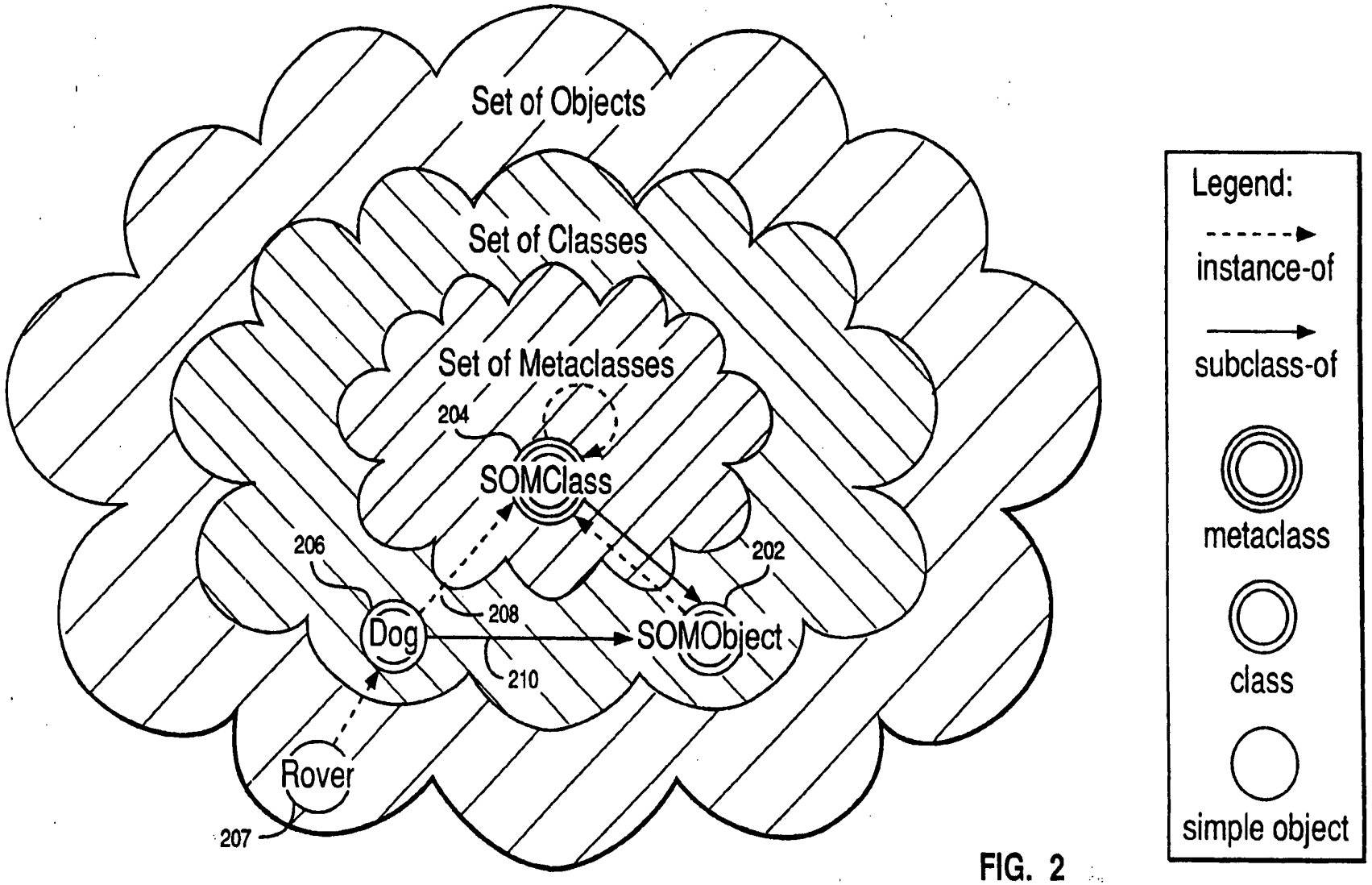

FIG. 2

PRIOR ART

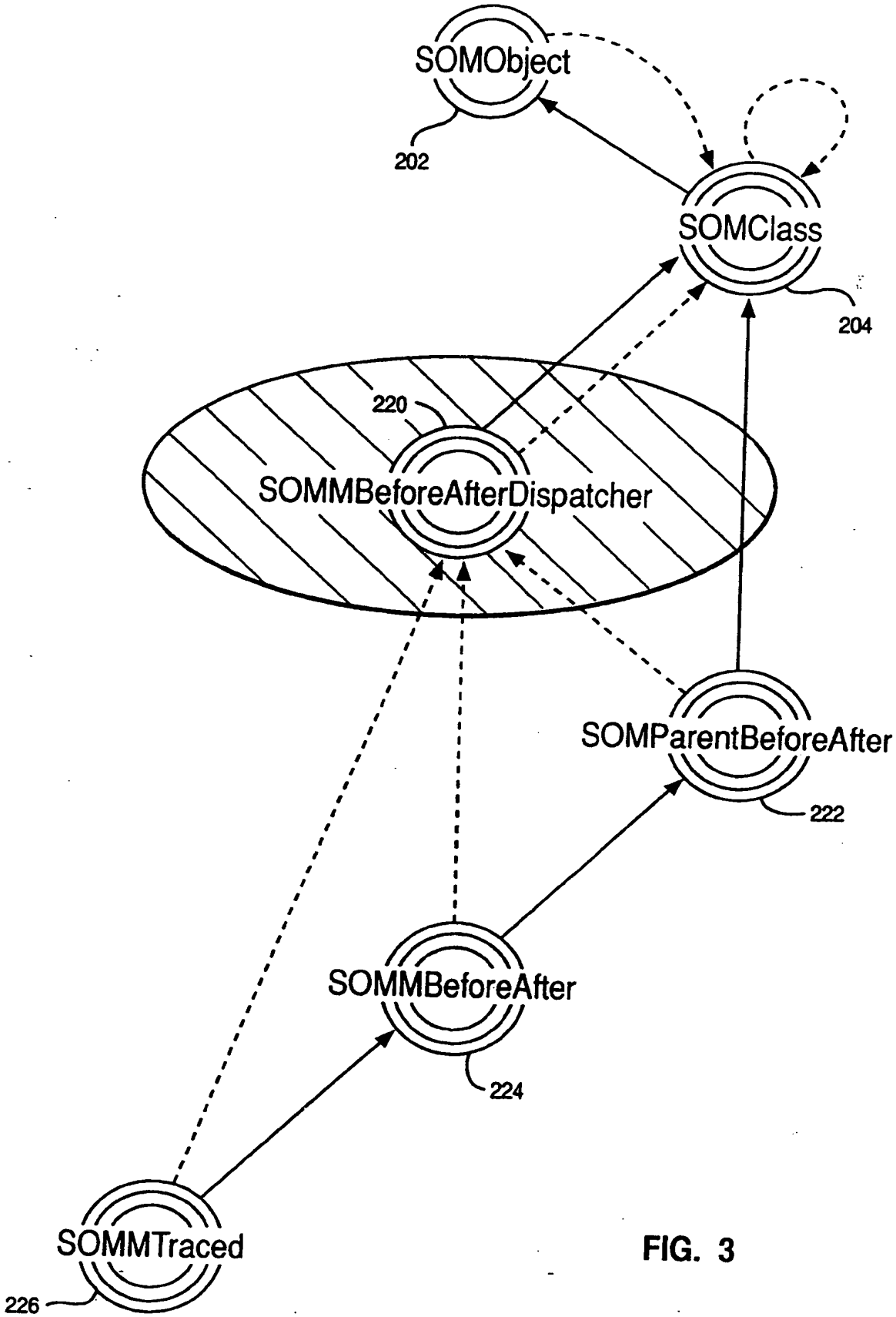
SOMObject
202
SOMClass
204
220
SOMMBeforeAfterDispatcher
SOMParentBeforeAfter
222
SOMMBeforeAfter
224
SOMMTraced
226

FIG. 3

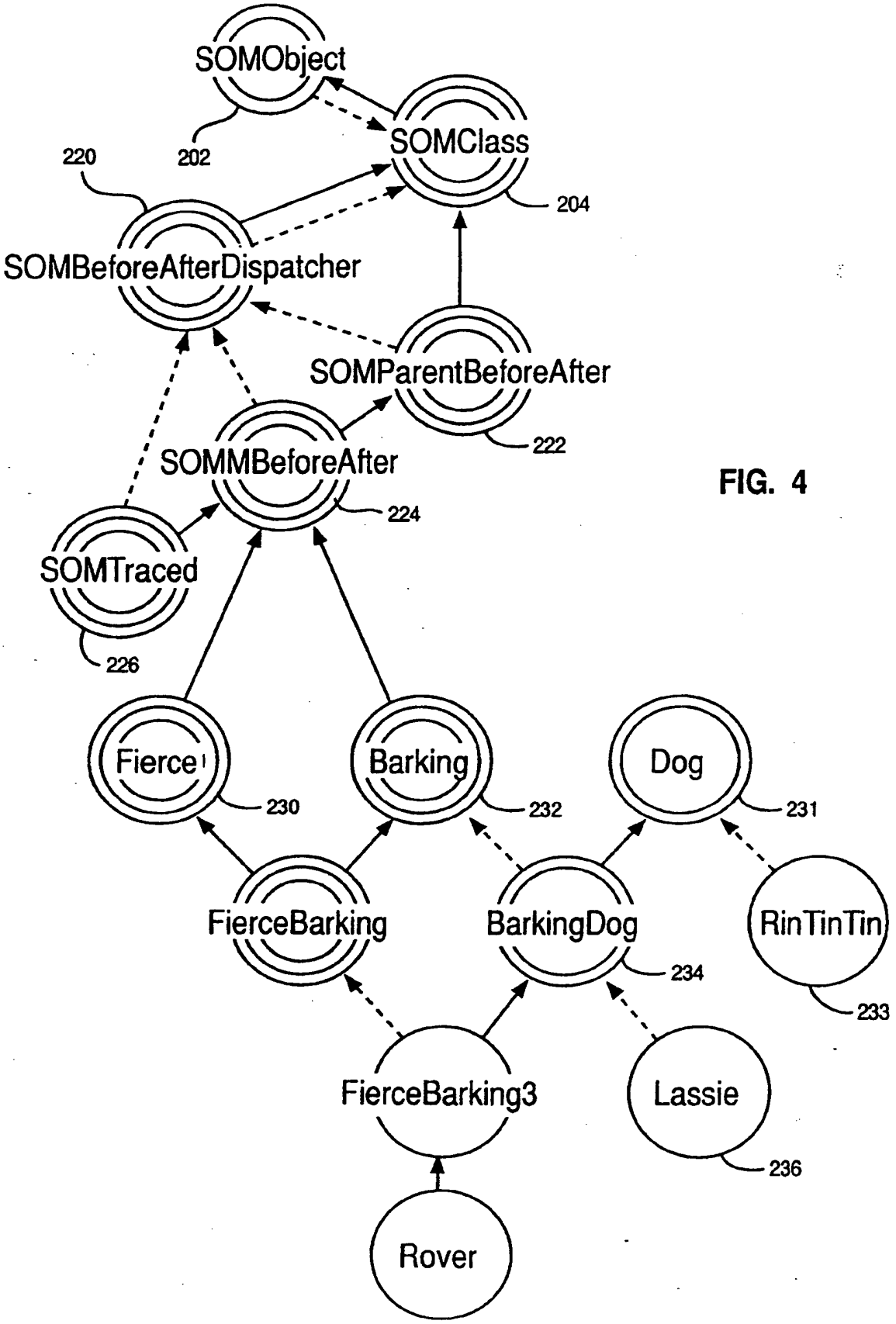
SOMObject
202
SOMClass
204
220
SOMBeforeAfterDispatcher
SOMParentBeforeAfter
222
SOMMBeforeAfter
224
SOMTraced
226
Fierce
230
Barking
232
Dog
231
FierceBarking
BarkingDog
234
RinTinTin
233
FierceBarking3
Lassie
236
Rover

FIG. 4

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 629 946 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **94304250.7**

(51) Int. Cl.[5] : **G06F 9/44**

(22) Date of filing : **13.06.94**

(30) Priority : **14.06.93 US 77229**

(43) Date of publication of application :
21.12.94 Bulletin 94/51

(84) Designated Contracting States :
DE FR GB

(88) Date of deferred publication of search report :
08.02.95 Bulletin 95/06

(71) Applicant : **INTERNATIONAL BUSINESS MACHINES CORPORATION**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : **Danforth, Scott Harrison**
10011 Woodland Village Drive
Austin, Texas 78750 (US)
Inventor : **Forman, Ira Richard**
2100 Cypress Point East
Austin, Texas 78746 (US)
Inventor : **Madduri, Hari Haranath**
7004 Anagua
Austin, Texas 78750 (US)

(74) Representative : **Davies, Simon Robert**
I B M
UK Intellectual Property Department
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(54) System and method for enabling before/after method processing in an object oriented system.

(57) A system for creating before and after behaviour upon invocation of a method in an object-oriented system. The framework provides metaclasses classes containing methods for dispatching a before method and an after method at the time of invocation of each client method in subclass instances. Object-oriented system properties of inheritance and encapsulation are supported as are derived metaclasses. Derivation ensures that the specification syntax for each class does not impact the expected result. The combination of explicit before after classes, dispatcher class, and derived metaclasses ensures that the system will have associative composition.

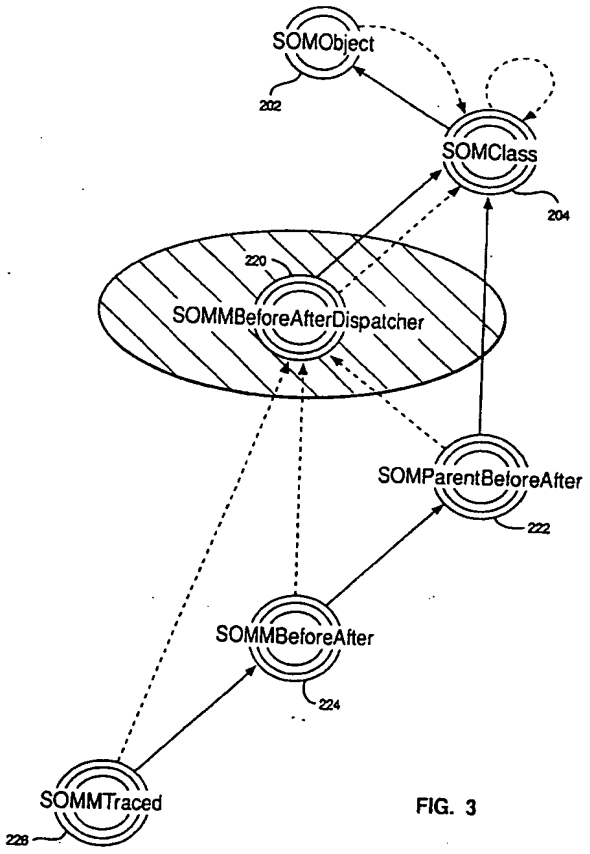

FIG. 3

EP 0 629 946 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 94 30 4250

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.5)
A	1990 INTERNATIONAL CONFERENCE ON COMPUTER LANGUAGES March 1990 , NEW ORLEANS, USA pages 190 - 197 XP289135 HARRISON AND OSSHER 'Subdivided Procedures: A Language Extension Supporting Extensible Programming' * the whole document * ---	1-3	G06F9/44
A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 35, no. 5 , October 1992 , NEW YORK US pages 337 - 338 'Resolution Procedure for Object Oriented Programming Systems.' * the whole document * ---	1-3	
A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 36, no. 3 , March 1993 , NEW YORK US pages 329 - 332 'Efficient Implementation of ACLS for Object-Oriented Systems.' * the whole document * -----	1-3	TECHNICAL FIELDS SEARCHED (Int.Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	6 December 1994	BRANDT, J

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)